MODAL - REPORT

Artgan Implementation

June 5, 2020

FORTES MACHADO Alicia, MARTINELLI LOPES Iago
X2018

CONTENTS

1 INTRODUCTION

In this project, we have implemented a Generative Adversarial Networks (GANs) [1], namely as ARTGAN, based on the paper where it was published. ARTGAN proposes an extension to normal GANs in order to synthesize more challenging and complex images, and the main innovation is to allow the back propagation of the loss function related to labels (assigned to each generated images) to the generator from the discriminator. We have successfully implemented ARTGAN, generated some images which are related to wikiart dataset [2] and to cifar-10 dataset, and we will be showing some of them in this report. We also faced some overfit problem which was not explained in the paper, we will be presenting two ways that we found in order to avoid this problem.

2 ARTGAN

2.1 Theory

The ARTGAN structure is similar to any other GANs, it has a generator that tries to fool a discriminator, while the discriminator tries to separate the images that were artificial generated from the real ones. Artgan innovation is to allow back-propagation from the label given to each generated image, so it is different than others GANs because both D and G receive an extra vector $\hat{y}$ that contains the label of the image.

As we can see from ARTGAN architecture (Figure 1), G receives a noise vector $Z \sim \mathcal{N}(0,1)^n$ and a vector $\hat{y}$ that contains the label of the image which will be generated, so it has size of K class, all zero except the label which is 1. G will generate an image through the Decoder using some deconvolutions layers. After, the generated image will pass through D and the decoder will try to separate into $K+1$ classes, with the class $K+1$ being the fake one. The discriminator will also receive a real image to be trained, and it gives a feedback to the Generator, which will try to generate a similar image. In D, we try to minimise the following loss related to the parameter θ_D.

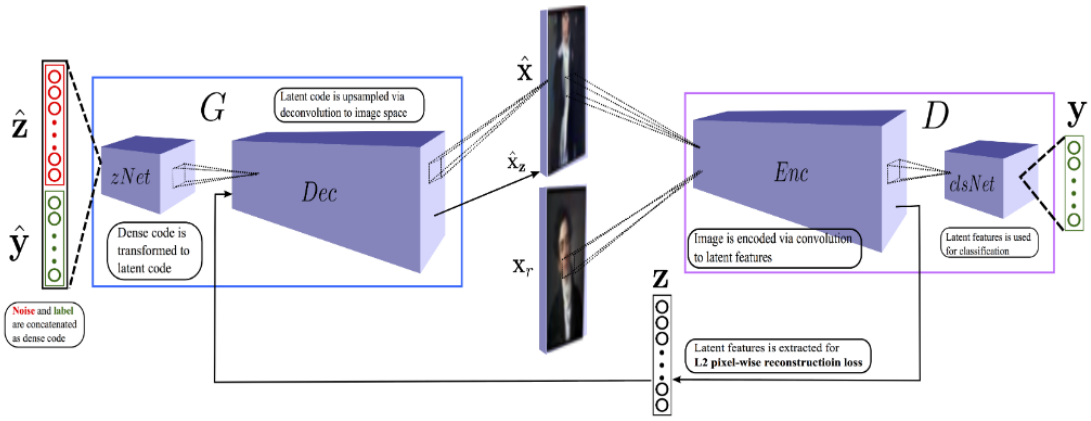

Figure 1: The overall architecture of ARTGAN.

$$\begin{aligned}\mathcal{L}_D = &- \mathbb{E}_{(x_r,k)\sim pdata}[\log p(y_i|x_r, i=k) \\ &+ \log(1-p(y_i|x_r, i\neq k))] \\ &- \mathbb{E}_{\hat{z}\sim pnoise,\hat{k}\sim K}[\log(1-p(y_i|G(\hat{z},\hat{y}_{\hat{k}}), i<K+1)) \\ &+ \log p(y_i|G(\hat{z},\hat{y}_{\hat{k}}), i=K+1)]\end{aligned} \tag{1}$$

Where x_r is a real image from the dataset, k is its label, $\hat{k}$ is the label of the input vector $\hat{y}$, and $\hat{z}$ is the input noise vector. Meanwhile, in G, we will be trying to maximise the loss $\mathcal{L}_D$ in order to update θ_G. Hence, the problem can be reformulated to the minimisation of the following loss.

$$\begin{aligned}\mathcal{L}_{adv} = &- \mathbb{E}_{\hat{z}\sim pnoise,\hat{k}\sim K}[(y_i|G(\hat{z},\hat{y}_{\hat{k}}), i=\hat{k}) \\ &+ \log(1-p(y_i|G(\hat{z},\hat{y}_{\hat{k}}), i\neq\hat{k}))]\end{aligned} \tag{2}$$

In order to improve training, the paper also suggests a L2 pixel-wise reconstruction loss, which could be used to update D and G, but we will be using this loss only to update θ_G with the following loss

$$\mathcal{L}_{L2} = \mathbb{E}_{x_r\sim pdata}[\left\|Dec(Enc(x_r)) - x_r\right\|_2^2] \tag{3}$$

The final form of the loss function for G is $\mathcal{L}_G = \mathcal{L}_{adv} + \mathcal{L}_{L2}$

2.2 Implementation

The paper specifications were clear enough that we could implement it without many difficulties. Its architecture, losses functions and optimizer were given, some hyperparameters such as batch size and optimizer values that were used by the authors were also given.

The first parameter that we had to choose by ourselves was the size of the noise vector $\hat{z}$, which after a brief research, we found out that the most used value is 100, so we also used it as the size of our noise vector. The second parameter that we had to decide was the size of the generated image, and this size would also be used in our transformation in the real dataset. After analysing ARTGAN architecture, we decided that 64×64 was the most appropriate size to our problem.

After everything was set, we decided to test our ARTGAN implementation with CIFAR-10 dataset, and we faced a huge overfitting problem, where after 20 epochs, the discriminator loss decreased to zero really quickly and the generator loss started to skyrocket. We concluded that the discriminator was becoming so smart that any attempt by the generator to trick it was failing. We had 2 approaches to this problem, and we propose a third one that we did not implement because it would go against what was specified in the paper. The first approach was to stop the training a moment before overfitting, the second one was to difficult the discriminator training by adding some noise to the real images that decayed over time. Those two approachs helped a lot to avoid overfitting, that happened around 40 epochs with CIFAR-10 dataset, and improved the quality of the generated images. A third approach to avoid overfitting would be to reduce the security of labels to less than 1, for example, 0.9. This would help because, in GANs, to be absolutely sure that a certain image belongs to a class greatly harms the quality of the generated images and leads to overfitting.

After this initial test, we trained our ARTGAN using wikiart dataset. Wikiart dataset is separated in three different types of classification, the first one is the genre, the second is the artist and the third one is the style. We trained using those three different types and it took more than a week to get 100 epochs inmaick each one using Microsoft Azure.

3

RESULTS

3.1 CIFAR-10

First, we trained with CIFAR-10, as the article indicates, in order to test if the net was working and the quality of its results. We can see the losses in Figure 2 and the results generated by the Generator in Figure 3.

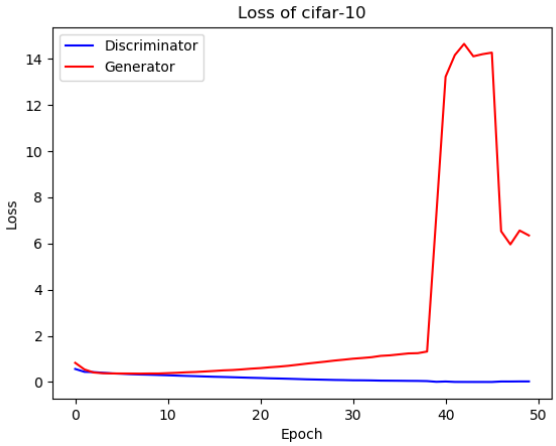

Figure 2: Loss for CIFAR-10

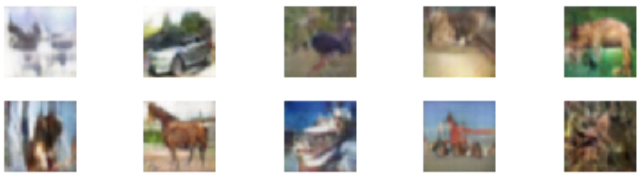

Figure 3: Grid for CIFAR-10

We can observe that at a certain moment the loss of the Generator increases abruptly. We hypothesize that this increase corresponds to when the Discriminator became much better than the Generator - possibly due to overfitting - which meant that the Generator could not improve anymore. Therefore, the best neural network, which was chosen to produce the images in Figure 3, was exactly the one before the loss explosion.

The images produced recall the elements of CIFAR-10, mainly in terms of shapes and colors. It is therefore possible to recognize the classifications. Those results resemble the ones produced in the article, but with less details.

3.2 WIKIART

For training the net to Wikiart, we made some changes to the code and created a class for this data set in order to read it. Thus, we were able to train the network and achieve the results presented below for the classes artist, style and genre.

We chose to train all classes in 100 epochs, since this quantity was the one described in the paper.

3.2.1 • Artist

The total time used for training this class (100 epochs) was nearly 2 days.

As we can see in Figure 4, we didn't experience an explosion in the loss as in CIFAR-10, what lead us to good results with the final neural network. Those results are shown in Figure 5. In this grid, the first line is with fake images that have good quality. The second line corresponds to images that were capable of fooling the Discriminator, even when he was very good.

About the results, we can see that the network was capable to learn important features as the background, clouds, buildings and general structure of a face. However, even if it learns well these concepts, we can see that they are still a little blurred.

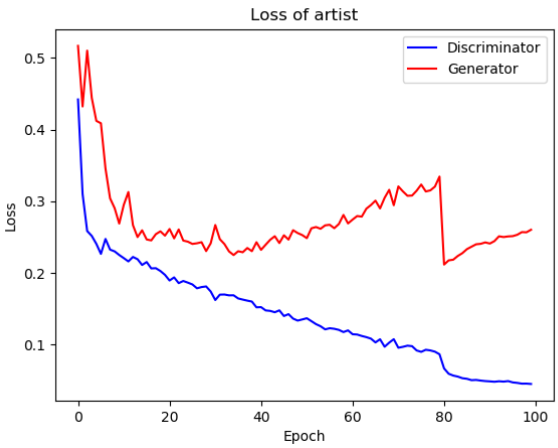

Figure 4: Loss for artist class

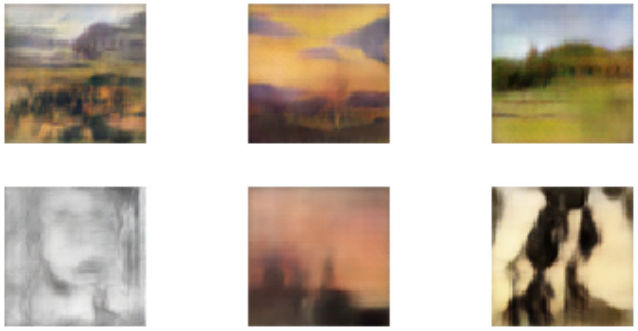

Figure 5: Grid for artist class. These images, from right to left and up to down correspond to Martiros Saryan, Nicholas Roerich, Camille Pissarro, Albrecht Durer, Ivan Aivazovsky and Vincent van Gogh

3.2.2 • Style

In this case, it took much more time to train, since the style class consider the whole data set and not partially. Also it has more classifications than the other classes. The total time was near 4 days.

We can see that the generator increased a lot its loss, which lead to images that were very blurred at the end. Thus, analysing our saved networks, we realized that the one that produced best images was the one correspondent to epoch 19.

3.2.3 • Genre

For the genre, it took a little less than the time used for training the artist class, since it has much less classifications - only 10. We can see in Figure 8 the losses for the Discriminator and Generator. We realized that there is a explosion near epoch 62. Thus a neural network before the explosion got the best results, which can be seen in Figure 9. The first column corresponds

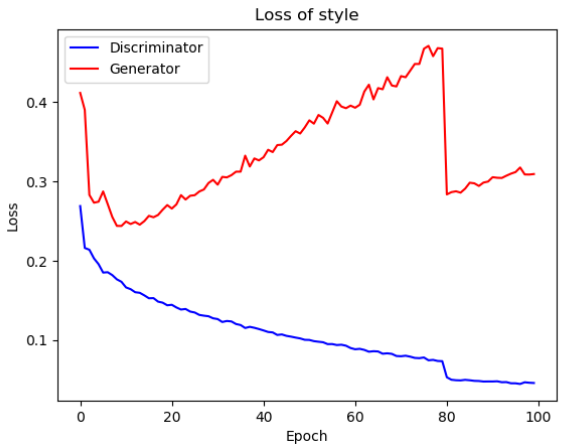

Figure 6: Loss for style class

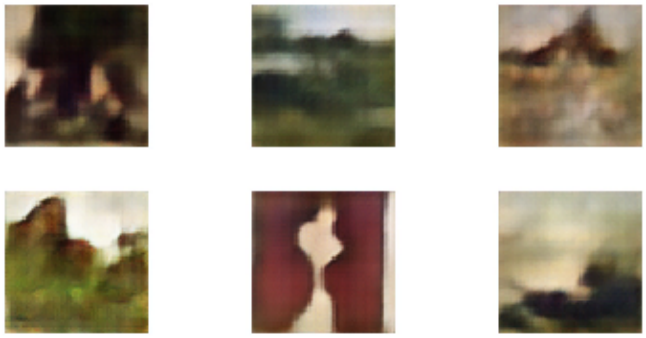

Figure 7: Grid for style class. These images, from right to left and up to down correspond to Baroque, Contemporary Realism, Impressionism, Naive Art Primitivism, Pop Art and Romanticism

to examples of portraits produced, the second one to cityscape samples and the third one to landscape samples.

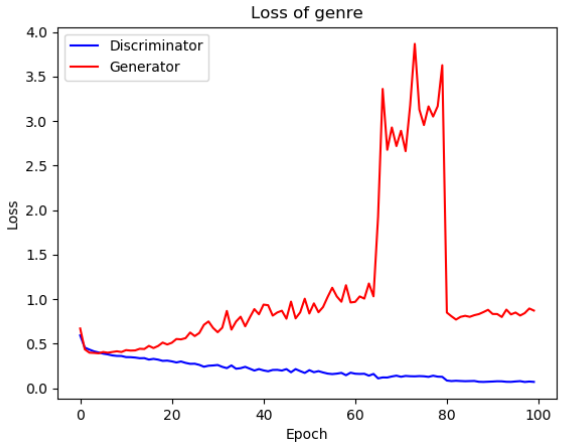

Figure 8: Loss for genre class

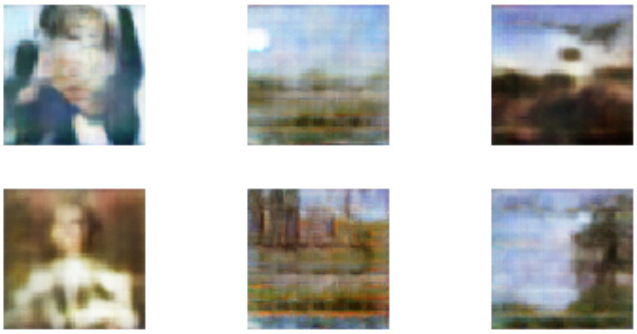

Figure 9: Grid for genre class. These images, from right to left, the columns correspond to Portrait, Cityscape and Landscape

4 CONCLUSION

Thus, we were able to replicate the results of the articles partially - we didn't get to the quality of the paper, but we produced good results. The fact that we did partially is probably due to unspecified implementation details. We used techniques used for GANs such as adding noise

at the beginning and gradually decreasing it and using the neural nets produced just before the Discriminator overfitting. We were able to train the network with CIFAR-10 and Wikiart, using colab and azure respectively.

REFERENCES

[1] Wei Ren Tan; Chee Seng Chan; Herńan E. Aguirre and Kiyoshi Tanaka. "ARTGAN: ARTWORK SYNTHESIS WITH CONDITIONAL CATEGORICAL GANS". In: (Apr. 2017).

[2] URL: https://github.com/cs-chan/ArtGAN/tree/master/WikiArt%20Dataset (visited on 05/11/2020).